Tutoriat 9 - Mass Storage Structure

Contents

1 Error Detection and Correction **6**
- 1.1 RAM . . . 6
- 1.2 HDD/NVM . . . 6

2 Storage Device Management **6**
- 2.1 Drive Formatting, Partitions, and Volumes . . . 6
- 2.2 Boot Block . . . 7
- 2.3 Bad blocks . . . 7

3 Swap Space Management **8**

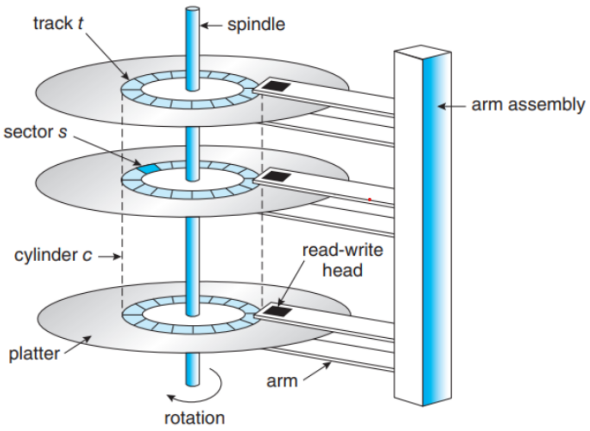
4 Storage attachment **8**
- 4.1 Host-attached storage . . . 8
- 4.2 Network-attached storage . . . 8
- 4.3 Cloud storage . . . 9
- 4.4 Storage-area networks and storage arrays . . . 9

Hard disk drives

Conceptually, HDDs are relatively simple. Each disk platter has a flat circular shape, like a CD. Common platter diameters range from 1.8 to 3.5 inches. The two surfaces of a platter are covered with a magnetic material.

We store information by recording it magnetically on the platters, and we read information by detecting the magnetic pattern on the platters. A read–write head "flies" just above each surface of every platter. The heads are attached to a disk arm that moves all the heads as a unit.

The surface of a platter is logically divided into circular tracks, which are subdivided into sectors. The set of tracks at a given arm position make up a cylinder. There may be thousands of concentric cylinders in a disk drive, and each track may contain hundreds of sectors.



One important performance aspect, the positioning time, or random-access time, consists of two parts: the time necessary to move the disk arm to the desired cylinder, called the seek time, and the time necessary for the desired sector to rotate to the disk head, called the rotational latency. Typical disks can transfer tens to hundreds of megabytes of data per second, and they have seek times and rotational latencies of several milliseconds.

Nonvolatile Memory Devices

NVM devices can be more reliable than HDDs because they have no moving parts and can be faster because they have no seek time or rotational latency. In addition, they consume less power. On the negative side, they are more expensive per megabyte than traditional hard disks.

NAND flash memory

In this scenario, we will be talking about SSDs that use nand flash memory to store data. NAND semiconductors have some characteristics that present their own storage and reliability challenges.

For example, they can be read and written in a "page" increment (similar to a sector), but data cannot be overwritten— rather, the NAND cells have to be erased first. The erasure, which occurs in a "block" increment that is several pages in size, takes much more time than a read (the fastest operation) or a write (slower than read, but much faster than erase).

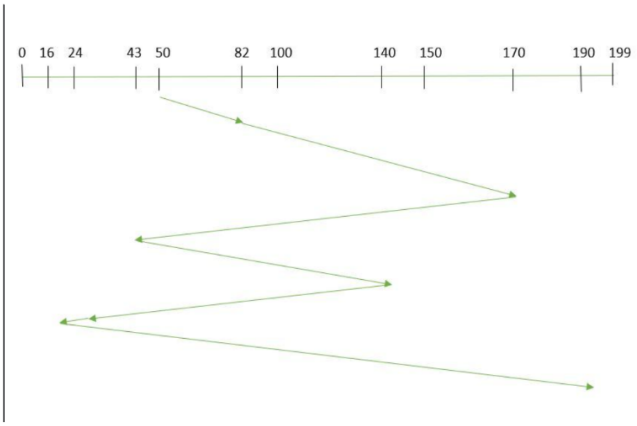
Because NAND semiconductors cannot be overwritten once written, there are usually pages containing invalid data. Consider a file-system block, written once and then later written again. If no erase has occurred in the meantime, the page first written has the old data, which are now invalid, and the second page has the current, good version of the block.

HDD Scheduling

As in the case of scheduling processes, the same thing happens for HDDs, which need to perform operations dictated by the cpu (for example retrieve data/write data onto itself).

FCFS (First-come first-served)

Let's suppose the order of request is $(82, 170, 43, 140, 24, 16, 190)$ and the current position of Read-Write head is at 50. The FCFS will serve the requests as follows:

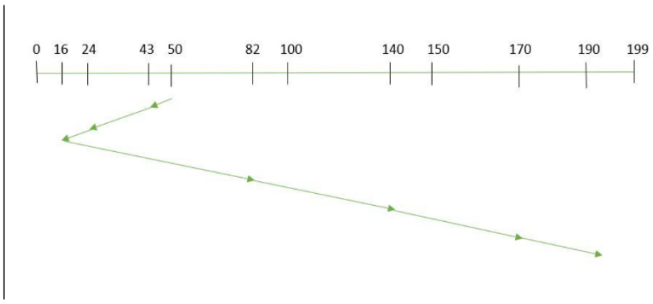


Total distance covered by the disk arm =

$$(82-50)+(170-82)+(170-43)+(140-43)+(140-24)+(24-16)+(190-16)=642$$

SSTF (Shortest Seek Time First)

The seek time of every request is calculated in advance in the queue and then they are scheduled according to their calculated seek time. As a result, the request near the disk arm will get executed first. Suppose the order of request is $(82, 170, 43, 140, 24, 16, 190)$ and current position of Read-Write head is at 50.

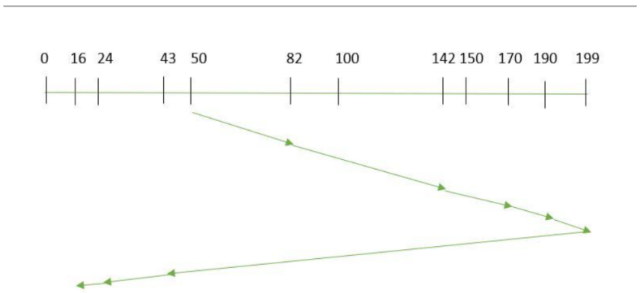


Total distance covered by the disk arm =

$$(50-43)+(43-24)+(24-16)+(82-16)+(140-82)+(170-140)+(190-170)=208$$

SCAN

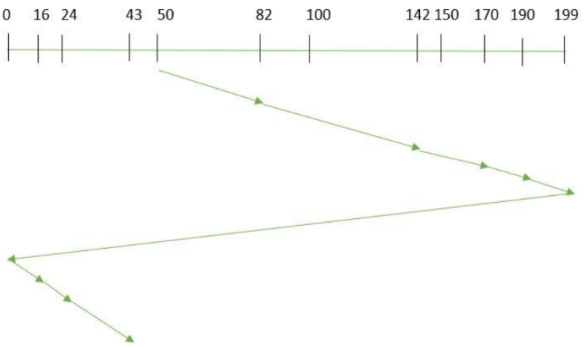
Here, the Read-Write head acts as a pendulum, going from one side of the disk to the start to the end of the disk, serving every request in its way. When it gets to the end of the disk it reverses its direction and again services the request arriving in its path. Suppose the request is $(82, 170, 43, 140, 24, 16, 190)$, the Read-Write arm is at 50, and the arm previously served request 35.



Total distance covered by the disk arm = $(199-50)+(199-16)=332$.

C-SCAN (Circular SCAN)

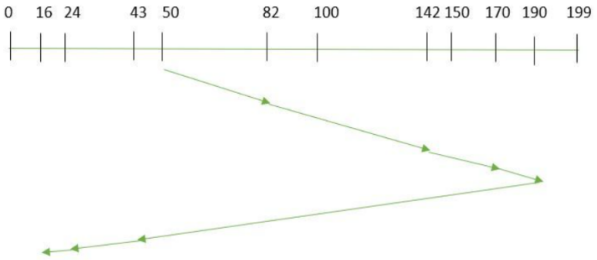
Like SCAN, C-SCAN moves the head from one end of the disk to the other, servicing requests along the way. When the head reaches the other end, however, it immediately returns to the beginning of the disk without servicing any requests on the return trip. Suppose the request is $(82, 170, 43, 140, 24, 16, 190)$, the Read-Write arm is at 50, and the arm previously served request 35.



Total distance covered by the disk arm $= (199 - 50) + (199 - 0) + (43 - 0) = 391$.

LOOK

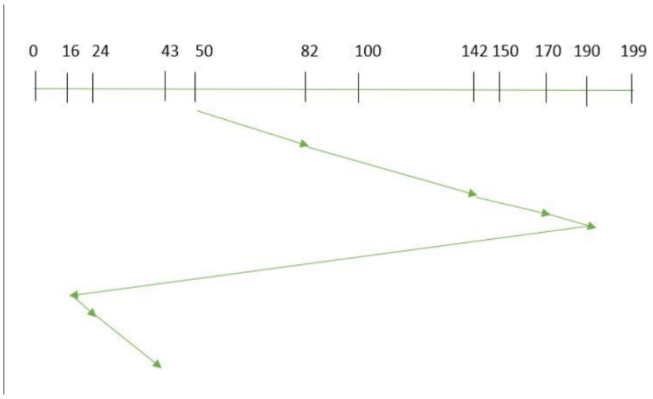
LOOK Algorithm is similar to the SCAN disk scheduling algorithm except for the difference that the disk arm in spite of going to the end of the disk goes only to the last request to be serviced in front of the head and then reverses its direction from there only. Thus it prevents the extra delay which occurred due to unnecessary traversal to the end of the disk. Suppose the request is $(82, 170, 43, 140, 24, 16, 190)$, the Read-Write arm is at 50, and the arm previously served request 35.



Total distance covered by the disk arm $= (190 - 50) + (190 - 16) = 314$.

C-LOOK

In C-LOOK, the disk arm in spite of going to the end goes only to the last request to be serviced in front of the head and then from there goes to the other end's last request. Thus, it also prevents the extra delay which occurred due to unnecessary traversal to the end of the disk.



Total distance covered by the disk arm = $(190 - 50) + (190 - 16) + (43 - 16) = 341$.

NVM Scheduling

The disk-scheduling algorithms just discussed apply to mechanical platter-based storage like HDDs. They focus primarily on minimizing the amount of disk head movement.

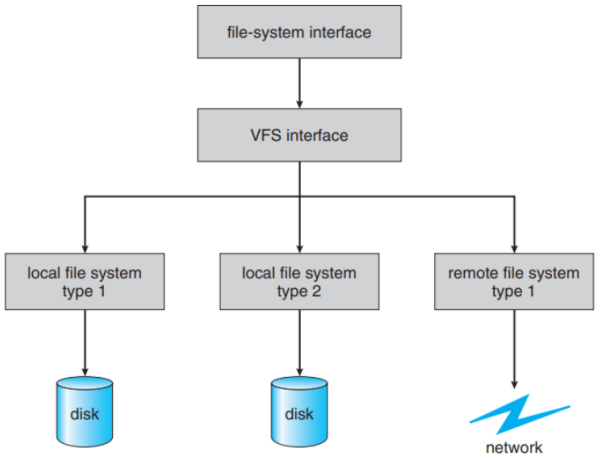
NVM devices do not contain moving disk heads and commonly use a simple FCFS policy. For example, the Linux NOOP scheduler uses an FCFS policy but modifies it to merge adjacent requests.

The observed behavior of NVM devices indicates that the time required to service reads is uniform but that, because of the properties of flash memory, write service time is not uniform. Some SSD schedulers have exploited this property and merge only adjacent write requests, servicing all read requests in FCFS order.

Virtual file system

As we've seen, modern operating systems must concurrently support multiple types of file systems. But how does an operating system allow multiple types of file systems to be integrated into a directory structure? And how can users seamlessly move between file-system types as they navigate the file-system space?

Most operating systems, including UNIX, use object-oriented techniques to simplify, organize, and modularize the implementation. The use of these methods allows very dissimilar file-system types to be implemented within the same structure, including network file systems, such as NFS. Users can access files contained within multiple file systems on the local drive or even on file systems available across the network.



The second layer is called the virtual file system (VFS) layer. The VFS layer serves two important functions:

1. It separates file-system-generic operations from their implementation by defining a clean VFS interface. Several implementations for the VFS interface may coexist on the same machine, allowing transparent access to different types of file systems mounted locally.

2. It provides a mechanism for uniquely representing a file throughout a network. The VFS is based on a file-representation structure, called a vnode, that contains a numerical designator for a network-wide unique file. (UNIX inodes are unique within only a single file system.) This network-wide uniqueness is required for support of network file systems.

The kernel maintains one vnode structure for each active node (file or directory). Thus, the VFS distinguishes local files from remote ones, and local files are further distinguished according to their file-system types.

1 Error Detection and Correction

Hardware is not perfect and sometimes bits get flipped because of too much usage, noise or other factors. Memory and storage handle this problems differently, here is how:

1.1 RAM

Parity bits

- Implemented at the hardware level - An extra bit that tracks the parity is added to every byte (ex: 1001 would actually be stored as 10010, where the last bit is the parity bit). - Detects 1 bit flips but fails for 2 or more. It also fails to provide a way to recover, for that we need an extended version called Hamming codes.

Hamming Codes

Hamming codes are a type of error correction codes (ECC) which provide both detection and recovery.

Multiple parity bits are used to also track the location of the changed bit. Here is an example:

1 0 1 1 would actually get stored as

0 1 1 0 0 1 1, where bits 1, 2, 4 are the parity bits.

Bit 1 detects changes in odd index bits (1, 3, 5, 7), Bit 2 would detect changes in 2, 3, 6, 7 and bit 4 for 4, 5, 6, 7.

Suppose bit 3 flips, now we have 0 1 0 0 0 1 1. This would invalidate parity bits 1 and 2 which tells us that bit 3 flipped.

Still can't detect 2 bits changes, for that we can add another global parity bit.

1.2 HDD/NVM

- Stores ECC per sector, updated on every write and checked on every read - When recoverable the system reports a soft error, otherwise a hard error. - Techniques for handling such situations are also described in the bad blocks section

2 Storage Device Management

2.1 Drive Formatting, Partitions, and Volumes

Low Level Formatting

- Takes disk from a piece of metal to something more usable by the OS - Split the disk into sectors - Fill each sector with a data structure that is made of sections (header, data and trailer) - Header and trailer hold metadata such as sector number and ECC - Done as part of the manufacturing process

//TODO: Discuss about different block sizes

- Low Level Formating is not enough to interact with files, the operating system needs to record it's own data structures on the device

Recording OS Structures

Partitioning:

- The OS logically splits the disk into partitions which is just an array of contiguous blocks - Each partition might hold different types of data (one holds the kernel executable, one the user files, one the swap space) - Information such as the start block and size of a partition is hold in a partition table

//TODO: Discuss why partitioning even exists

Volume Creation:

- Implicit when a filesystem is placed directly in a partition - Another layer of abstraction over partitions that offers more flexibility - A volume can span multiple disks and can be resized dynamically (unlike partitions)

Logical Formatting:

- Initialization of the filesystem data structures (empty root directory, maps of free/allocated blocks)

2.2 Boot Block

The boot process is made of the following steps:

- A simple bootstrap program runs and instructs the storage controller to load the (boot) blocks containing a more complicated bootstrap program in memory. This program is usually embedded in non volatile memory (NVM) by the manufacturers.

- The complicated bootstrap program scans the disk for the operating system and loads it into memory (or just reads the OS from e predefined location).

Windows boot process

- The firmware loads (in memory) the fist block from the disk (called the master boot record, MBR). - The MBR holds the code that loads the OS and the partition table. - One entry of the partition table holds the OS and it is called the boot partition - The boot code from the MBR then starts executing the code in the first sector of the boot partition called the boot sector.

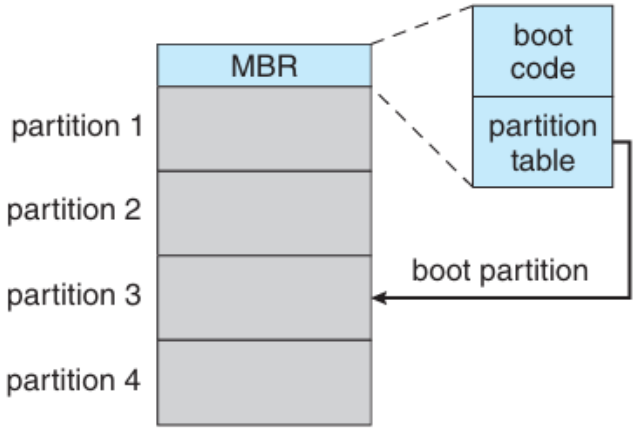


Figure 1: Master Boot record

2.3 Bad blocks

- Some of the disks sectors might get corrupted (or be already corrupted before using it). - Detection is done with scanning and ECC. - "Recovery" can be done in multiple ways:

Sector Sparing

Keep lists for bad and spare blocks and redirect requests for bad to spare.

Example:

• The operating system tries to read logical block 87.

• The controller calculates the ECC and finds that the sector is bad. It reports this finding to the operating system as an I/O error.

• The device controller replaces the bad sector with a spare.

• After that, whenever the system requests logical block 87, the request is translated into the replacement sector's address by the controller.

Sector slipping

Move all the sectors after the bad block with one step until a spare sector is found.

Example:

Suppose that logical block 17 becomes defective and that the first available spare sector is sector 202. Sector slipping remaps all sectors from 18 through 202 by shifting them one position forward. Specifically, sector 202 is copied into the spare sector, sector 201 is copied into sector 202, sector 200 into sector 201, and so on, until sector 18 is copied into sector 19. This process frees sector 18, allowing logical block 17 to be mapped to it.this way frees up the space of sector 18 so that sector 17 can be mapped to it.

3 Swap Space Management

When there is no more space in memory, part of the data can be temporarily moved to the disk to make space. The special disk place is called swap space.

There are multiple ways of handling swap space, whole process images could be moved or only parts of it (like it's heap or memory pages that don't get accessed frequently).

Another problem that might arise is how much disk is used for swapping. //TODO: Talk about advantages/disadvantages

Some operating systems—including Linux—allow the use of multiple swap spaces, including both files and dedicated swap partitions.

Swap space can be part of the filesystem (it becomes a normal, but big, file) or live in a separate partition called a raw partition (with no filesystem and no directory structure).

4 Storage attachment

Computers access secondary storage in three ways: via host-attached storage, network-attached storage, and cloud storage.

4.1 Host-attached storage

Any type of storage that talks to a computer through local I/O ports (rather than using something like a remote server) is known as **host-attached storage**. These ports use several technologies, the most common being SATA. A typical system has one or a few SATA ports.

High-end systems and servers generally need more storage, so they use more sophisticated I/O architectures, such as **fibre channel (FC)**. This allows a server to access storage that might be in a different room or even building, as if it were a local drive.

A wide variety of storage devices are suitable for use as host-attached storage, such as HDDs, NVMs, CDs, DVDs, Blu-rays, tape drives, and **SANs (storage-area networks)**.

4.2 Network-attached storage

This method (**NAS**) provides a way to share storage across a network, rather than plugging it directly into a single computer. Unlike host-attached storage, which is private to one machine, a NAS is a centralized "pool" that many users can access at once.

Clients access NAS via a remote-procedure-call interface, such as NFS for Unix/Linux, or CIFS for Windows. The remote procedure calls (RPCs) are carried via TCP or UDP over an IP network - usually the same local-area network (LAN) that carries all traffic data to the clients.

iSCSi is the latest network-attached storage protocol. It sends standard storage commands over a regular IP network; while NFS and CIFS send files, iSCSi sends logical blocks across the network and leaves it to the client to

use the blocks directly or to create a file system with them. As a result, the computer thinks the storage is **directly attached**, even if it is distant from the host.

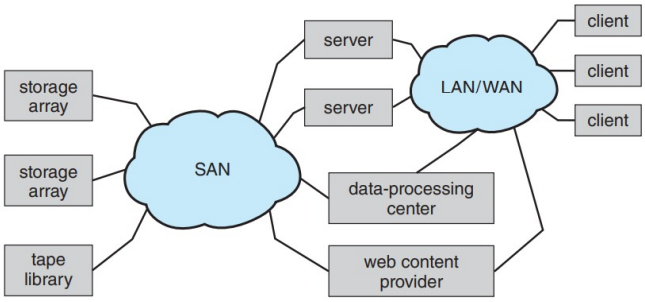


Figure 2: Network-attached storage

4.3 Cloud storage

Similarly to NAS, cloud storage provides access to storage across an network. However, storage is accessed across the Internet or another wide-area network (WAN) to a remote data center that provides storage for a fee (or even for free).

Another difference between NAS and cloud storage is that NAS is just accessed as another file system if the CIFS or NFS protocols are used, or as a raw block device if the iSCSi protocol is used, whereas cloud storage is API based; programs have to use APIs to access storage. Some examples would be Dropbox, Apple iCloud, and Microsoft OneDrive.

One reason that APIs are used instead of existing protocols is the latency and failure scenarios of a WAN. NAS protocols were designed for use in LANs, which have lower latency than WANs and are much less likely to lose connectivity between the storage user and the storage device. If a LAN connection fails, a system using NFS or CIFS might hang until it recovers. With cloud storage, failures like that are more likely, so an application simply pauses access until connectivity is restored.

4.4 Storage-area networks and storage arrays

Figure 3: Network-attached storage

One drawback of network-attached storage systems is that the storage I/O operations consume bandwidth on the data network, thereby increasing the latency of network communication. The communication between servers and clients competes for bandwidth with the communication among servers and storage devices.

A **storage array** is a purpose-built device that includes SAN ports, network ports, or both. It also contains drives to store data and a controller (or redundant set of controllers) to manage the storage and allow access to the storage across the networks. The controllers are composed of CPUs, memory, and software that implement the features of the array, which can include network protocols, user interfaces, RAID protection, snapshots, replication, compression, deduplication, and encryption.

Some storage arrays include SSDs. An array may contain only SSDs, resulting in maximum performance but smaller capacity, or may include a mix of SSDs and HDDs, with the array software (or the administrator) selecting the best medium for a given use or using the SSDs as a cache and HDDs as bulk storage.

A **storage-area network (SAN)** is a private network (using storage protocols rather than networking protocols) connecting servers and storage units. Multiple hosts and multiple storage arrays can attach to the same SAN, and storage can be dynamically allocated to hosts. The storage arrays can be RAID protected or unprotected drives (**JBOD = Just a Bunch Of Disks**). A SAN switch allows or prohibits access between the hosts and the storage. For example, if a host is running low on disk space, the SAN can be configured to allocate more storage to that host. SANs make it possible for clusters of servers to share the same storage and for storage arrays to include multiple direct host connections. SANs typically have more ports and cost more than storage arrays. SAN connectivity is over short distances and typically has no routing, so a NAS can have many more connected hosts than a SAN.

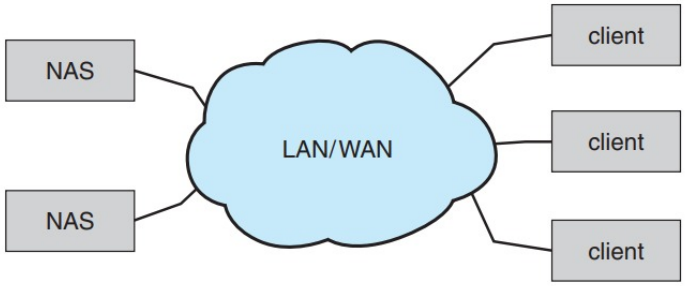


Figure 4: Network-attached storage